NPS-CS-07-012

NAVAL
POSTGRADUATE
SCHOOL

MONTEREY, CALIFORNIA

SecureCore Security Architecture: Authority Mode and Emergency Management

by

Timothy E. Levin, Ganesha Bhaskara, Thuy D. Nguyen, Paul C. Clark, Terry V. Benzel, Cynthia E. Irvine

16 October 2007

Prepared for: the National Science Foundation and the Defense Advanced Research Projects Agency

This page left intentionally blank

NAVAL POSTGRADUATE SCHOOL
Monterey, California 93943-5000

Daniel T. Oliver President	Leonard A. Ferrari Provost

This report was prepared for and funded by: NSF and DARPA.

Reproduction of all or part of this report is authorized.

This report was prepared by:

Timothy E. Levin
Research Associate Professor

Reviewed by:	Released by:
Peter J. Denning Department of Computer Science	Dan C. Boger Interim Associate Provost and Dean of Research

This page left intentionally blank

REPORT DOCUMENTATION PAGE	Form approved OMB No 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources,
gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	**2. REPORT DATE** 16 October 2007	**3. REPORT TYPE AND DATES COVERED** Research; 10/16/06 – 10/16/07

4. TITLE AND SUBTITLE SecureCore Security Architecture: Authority Mode and Emergency Management	**5. FUNDING** Grant numbers: CNS-0430566 and CNS-0430598
6. AUTHOR(S) Timothy E. Levin, Ganesha Bhaskara, Thuy D. Nguyen, Paul C. Clark, Terry V. Benzel, Cynthia E. Irvine	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Center for Information Systems Security Studies and Research (NPS CISR) 1411 Cunningham Rd., Monterey, CA 93943	**8. PERFORMING ORGANIZATION REPORT NUMBER** NPS-CS-07-012
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Science Foundation, 4201 Wilson Blvd. 1175 N. ArlingtonVA22230 DARPA, 3701 Fairfax Drive, Arlington, VA 22203	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES

The views expressed in this report are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited	**12b. DISTRIBUTION CODE**

13. ABSTRACT (Maximum 200 words.)

This document describes research in the SecureCore project. It describes results in the integration of SP Authority Mode hardware security features into the SecureCore Software Security Architecture and the SecureCore Trusted Management Layer. The context of integration is described including the target security policy, transient trust requirements and rationale. The design for protecting secondary storage volumes, trusted channels, and emergency management is described. Design considerations and recommendations for future work are also included.

14. SUBJECT TERMS Operating systems: Security and protection: separation kernels, access controls, multilevel security Organization and design: hierarchical design Communications Management: network communication Process Management: multiprocessing/multiprogramming/multitasking Hardware: Register-transfer-level-implementation: design	**15. NUMBER OF PAGES** 19 **16. PRICE CODE**

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

This page left intentionally blank

NPS-CS-07-012

SecureCore

Trustworthy Commodity Computation and Communication

SecureCore Technical Report

SecureCore Security Architecture:

Authority Mode and Emergency Management

Timothy E. Levin, Ganesha Bhaskara, Thuy D. Nguyen, Paul C. Clark,
Terry V. Benzel, Cynthia E. Irvine

This page intentionally blank

Acknowledgement

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0430566 and CNS-0430598 with support from DARPA ATO. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation or of DARPA ATO.

Author Affiliations

Naval Postgraduate School:

Cynthia E. Irvine, Timothy E. Levin, Thuy D. Nguyen, and Paul C. Clark

Center for Information Systems Security Studies and Research
Computer Science Department
Naval Postgraduate School
Monterey, California 93943

USC Information Sciences Institute:

Terry V. Benzel and Ganesha Bhaskara

Information Sciences Institute
University of Southern California
4676 Admiralty Way, Suite 1001
Marina del Rey, Ca 90292

Forward

This document describes research in the SecureCore project. It is written for members of the SecureCore project and others who understand the logic of pseudo-code and who are aware of basic SecureCore design features, including SP Authority Mode.[5][9] The SecureCore Trusted Management Layer is being developed in phases.[2] This document is targeted at Phase II, and with the assumption that there is a single Third Party and one Emergency Partition. Generalizations to extend the concepts presented here, e.g., to support multiple Third Parties, are the objectives of future work.

Typographic conventions are used to distinguish *footnotes, citations, endnotes* and *pseudo-code.* Footnotes are marked with superscript Arabic numbers, the body of which should occur on or near the same page as the mark. Citations to other publications are marked numerically within square brackets, the bodies of which are collected at the end of the document. End notes are marked with superscript lower-case Roman numbers, the bodies of which are collected in the Appendix. Pseudo-code is formatted in courier font (`like this`).

TABLE OF CONTENTS

This page intentionally blank

I. Introduction

During many crises, first-responder access to sensitive, restricted emergency information is required. Lives can be saved and property protected during disasters if first responders can use various medical and government records, such as the addresses and identities of handicapped and susceptible people. Yet, currently, access to such information is restricted and is often not available for first responders during emergencies because government and third party data providers lack confidence in the ability of emergency IT systems to protect their data from unauthorized access. To be practical, a solution to this problem must be usable "in the field," and provide dual-use functions on familiar equipment. The target platform on which the SecureCore design fulfills these requirements is a handheld computer, referred to hereafter as the *SC_Device*.

SecureCore provides a mobile emergency-response capability that ensures third party information protection during emergencies, providing a trusted foundation for more effective crisis management activities. SecureCore is designed to enforce information security policies that are:

- Demonstrably assured,
- Persistent during emergency as well as non-emergency periods, and
- Global across all participating organizations

The foundation of the SecureCore solution[6], consists of two major components: *Authority Mode SP* hardware [5][9] and the *Trusted Management Layer* (TML). Authority Mode SP (i.e., SP, enhanced with Authority Mode) provides a basis for key management and protected communications. The *Trusted Management Layer* (i.e., Least Privilege Separation Kernel or LPSK, and SecureCore Security Services, or SCSS) partitions all platform resources and controls interactions between the resources.[8]

SecureCore Module and Partition Terminology

Active partitions are those that the TML schedules to receive a time slice, whereas *passive partition*s are never scheduled (e.g., may be used to hold data). The current generation of SecureCore (including all planned phases) is a single CPU platform in which all active partitions share the CPU, as controlled by the TML.

The *Trusted Partition* is an active partition that contains the *SecureCore Operating System*, which provides support for high assurance *trusted applications*, including the *Trusted Path Application* (TPA). The *Emergency Partition* is an active partition that is configured for the secure handling of sensitive data during a declared emergency situation. All partitions other than trusted and Emergency Partitions are called *Normal Partitions*. Normal, active partitions can host a commodity operating system (OS) or specialized applications.

SP supports Authority Mode Trusted Software Modules (*ATSM*s) and User Mode Trusted Software Module (*UTSM*s). SP does not limit which partition, process, or ring these modules are allocated to, or how many of them are configured to run on a given platform.

A. Business Model

SC_Devices are used to provide transient access to emergency information. SC_Devices are owned, maintained and operated by an organization we refer to as the *Authority*. The emergency information is owned by third party organizations. In Phase II, the SC_Device will communicate with a single *Third Party*, although the design is intended to be easily extendable to multiple Third Parties in later phases.

In many cases, there are financial, regulatory and political hurdles to sharing information both during an emergency and otherwise. The Authority establishes an *agreement* (e.g., MOU or SLA) with the Third Party regarding their various expected behaviors, such as the precise information sharing policies and level of protection that will be afforded to shared information.

The Authority and the Third Party may agree for the SC_Device to host one or more Third Party *trusted applications* in the Trusted Partition. The level of trustworthiness of these applications and the amount of trust that the Third Party allocates to them are outside of the scope of this document, but the security of the SC_Device does depend on those factors.

The Third Parties rely on the Authority to correctly configure the SC_Devices. The Third Parties also rely on the Authority to declare the start and end of the emergency in an appropriate and secure manner.

B. Operational Model

The Authority and individual Third Parties exchange symmetric-cryptography keying material to support communication channels with appropriate levels of confidentiality, integrity, authentication, and authorization. Additionally, SecureCore supports the following operations and features:

- The Authority sets up SC_Devices at a device-configuration *Depot,* to include installation of: cryptographic keys, Third Party programs and pre-distributed Third Party emergency data.
- The Authority manages the global *emergency state* (i.e., it is either on or off)[i], including secure communication of changes in the emergency state to SC_Devices and Third Parties.
- Authorized users can temporarily access certain sensitive information during emergencies, which they cannot access under normal conditions.
- The SC_Device manages secure communication of emergency data between SC_Devices in the field and Third Parties.
- The User, the Authority or the Third Party can revoke trust relationships between them during operation.
- SCOS provides a *trusted application* environment for high integrity operations such as secure signing of documents and secure session management.

See Appendix A for a description of anticipated extensions to these features.

II. Security Policy

A multilevel security (MLS) policy is enforced on the SC_Device, and is implemented by using mandatory access control labels. Each partition on the SC_Device is assigned a label or in the case of a *Trusted Partition*, a label range. This policy further allows the usual MLS restrictions on access to data to be bypassed during an emergency, within predefined limits:

- An authorized user of the SC_Device shall not be allowed to access the emergency data except during an emergency.
- The emergency data shall not be accessible by remotely connected users, other than those acting as the Authority or the authorized Third Party. Third Parties may remotely access their own emergency data during or outside of an emergency, e.g., to be able to maintain the platform-resident emergency data.

The agreement between the Authority and a Third Party will establish the label for the Third Party's *Normal Partition* (if there is such a partition) and for its trusted applications in the Trusted Partition, if any, and the label for the Third Party's *Emergency Partition*(s).

The communication of changes of emergency state to SC_Devices, and the local representation of emergency state on the devices, are protected with high integrity to ensure coherent emergency response.

SecureCore provides the capability for emergency state changes to be confidential, since different sorts of emergencies can require different forms of confidentiality. For example, consider these scenarios:

- A fire truck siren is meant to be public to ensure traffic safety and rapid response. In the announcement of an impending tsunami, it is critical that as many people as possible know about the emergency.
- There is no reason for the public to know the <u>nature</u> of the emergency when a fire truck drives by.
- Confidentiality of emergency state can shield first responders from social vultures (e.g., "ambulance chasers") or malicious parties who could intentionally interfere with emergency response.
- Military emergencies can involve highly sensitive national security information requiring strict confidentiality, such the wreckage from a "balloon-launched classified government project."[3]
- The network-based announcement of an emergency may include more than just the start of the emergency, depending on how the protocol is designed, and what the end users need to know.[ii]

Thus, in some configurations it would be necessary for the emergency state have both high integrity as well as non-zero confidentiality.

Consistency of local and global emergency state is important, although the natural variability of mobile device connectivity limits the degree of consistency that is achievable. When an SC_Device loses connectivity with the Authority during an emergency, it is allowable for the user to continue to have access to the device's local emergency information, as limited by a configurable timeout value, until connectivity is reestablished and the SC_Device can re-synchronize with the global emergency state.

The Authority is proxy for the Third Parties in terms of the security policy. No attempt is made to protect Third Party communications, data or code *from* the Authority, given that the Authority installs all cryptographic keys and trusted software.

A. Application Policies

Object-specific handling instructions (e.g., "application policies") may be cryptographically bound to certain objects, in addition to the objects' MLS labels that the TML may provide. These instructions can indicate restrictions on access to the objects, over and above the restrictions denoted by the MLS labels. The instructions can be interpreted by an A-TSM or UTSM, which would also be responsible for managing the objects in a manner consistent with the instructions. Articulation of feasible application policies and suitable architectural requirements to ensure the sufficiency of the trusted programs to enforce the instructions are outside of the scope of this document.

B. Cryptographic Policy

To support the MLS policy, the Crypto Policy places requirements on the use of cryptographic keys and algorithms relative to the security level of data, its processing, or its communication.

- The SC_Device must ensure that the decryption methods for inbound data are commensurate with the destination partition security level, even if the OS on the remote (source of data) platform is subverted (e.g., to prevent an UNCLASS partition from receiving decrypted data from classified source).
- The SC_Device must ensure that the encryption methods for outbound data are commensurate with the source partition security level, even if that partition's (untrusted) OS is malicious.

Cryptographic keying material is handled in different ways:

- The Authority shares a separate secret with each device, which is installed into the hardware at the Depot (i.e., the SP Device Root Key[5] or *DRK*). Third Parties may not access the DRK.
- The Authority shares a separate secret with each Third Party, which may not be shared with any of the SC_Devices.
- Third Parties may have a secret installed in TML-protected memory on certain SC_Devices, to support secure communication with the SC_Devices.

C. Devices

The MLS characteristics of SC_Device *physical I/O devices* can be managed in several different ways:

- Single level – I/O device has a fixed security level
- Single Level at a Time (SLAT) – I/O device security level varies over time, within a fixed range

- Multilevel – the I/O device has a range and IP packets are explicit or implicitly-labeled within that range

If partitions with different security levels are configured to utilize the same physical I/O device, then that device will need to change levels to accommodate the currently scheduled partition (viz., SLAT), or it will need to be multilevel.

The SC_Device can be configured so that security levels are bound to certain remote IP addresses. When network data is sent or received by a partition, the SC_Device must ensure that the remote-address level matches the partition level and matches or is within the range of the physical device.[iii]

III. Transient Trust: Requirements and Rationale

The TML creates and manages the Emergency Partition to provide users with emergency access to sensitive data. The user of the SC_Device may only access emergency data during emergencies, as described next.

Emergency data is kept in Emergency Partitions. A user in a Normal Partition can never access data in an Emergency Partition. Users can enter the Emergency Partition only during an emergency. Access by users to Emergency Partitions is protected by the TML, which has knowledge of the global emergency state, and the trusted path application, which is the gatekeeper to partitions. Processes in the Emergency Partition are not allowed to write to resources outside of that partition, except for writing to the Third Party via a trusted network channel.

Users can hop back and forth between emergency, trusted, and Normal Partitions during an emergency. Once an emergency is over, the TML and TPA make the Emergency Partition inaccessible to the user of the SC_Device.

The TML can be configured so that after the emergency, it copies a snapshot of emergency data to a static protected partition (e.g., for audit of generated or modified data), or this data can be transmitted to the Third Party for update of their databases.[iv]

Third parties can securely transmit data to the SC_device for storage in the Emergency Partition during or outside of an emergency. A daemon process in the Emergency Partition manages the reception of emergency data on the SC_device.[v]

IV. Design

Access to sensitive information during an emergency is based on the TML, which is comprised of the Least Privilege Separation Kernel (LPSK) and the SecureCore Security Services (SCSS) components. The LPSK exports various resource abstractions, organizes them into partitions, and enforces a security policy regarding information flow between partitions. The SCSS provides OS-support services and semantics associated with the LPSK-exported resources.

A. Subjects

In Phase II, there will be one process per partition. A multi-program process can be structured to utilize the several *rings* (i.e., hardware privilege levels) available outside of the TML, e.g., one program per ring. Thus, a “subject” is a process-ring pair.

B. Objects

The TML exports object abstractions: *memory segments*, and synchronization primitives (i.e., *eventcounts* and *sequencers*[10]); and two abstractions of object groups: *disk volumes,* and *segment volumes*. The TML labels all exported objects. The guest OS of one partition can mount the disk volume of another partition, as allowed by the TML security policy, which enables sharing of persistent data between the partitions. However, if the guest OS creates its own object abstractions within a volume, the TML cannot help enforce OS restrictions (e.g., for applying the principle of least privilege) over and above the restrictions of the TML security policy.

C. Segments and Segment Volumes

Segments are made available by the TML via a SecureCore-specific interface. Segments have the advantage that they can be individually protected by LPSK (as opposed to OS-created objects inside of a volume). Segments have a global name space that is subject to MLS restrictions. For example, the segment names and certain other static metadata may be labeled separately from the segment itself.[7] User-accessible segments are organized in *exported* segment volumes managed by the LPSK. Similarly, the TML maintains one or more *internal* segment volumes for its own use, (although its internal objects may not be labeled).

The segment volume structures (i.e., all of the segments and related metadata that comprise a segment volume) are encrypted using `sp_secure_store` from within an ATSM module of the TML. A general description of how TML secures segment volumes is provided in the following example (*sealing procedures* and *storage procedures*), and in Figure 1[1].

1. Sealing Procedures

Sealing procedures are used to generate and store cryptographic signatures of segment volumes. These volumes can be signed at various intervals or in response to different events such as a request to shut down the system.[vi] For this description, it is assumed that there is one exported and one internal volume (called `evol` and `ivol`), but extension to support multiple segment volumes should be straightforward.

TML calls an internal `ATSM` module with volume check-sum values it has generated to generate a seal. `_atsm_seal_vols` moves each check-sum to a register (`hw_move` is an abstraction of a typical assembly operator) and then calls `sp_gr_to_srh` to generate and store the seal (shown as V-hash in Figure 1) in the `SRH` register of Authority Mode SP.

```
_atsm_seal_vols (ivol-hash, evol-hash: word) = (
      hw_move(R1, evol-chksum)
      hw_move(R2, ivol-chksum)
      sp_gr_to_srh(R1, R2))
```

The TML design process will establish the details of how TML generates volume checksums.[2]

[1] Regarding pseudo-code: each function is prefixed to indicate the name of the component in which it resides. Internal functions of a component, which are not available at its interface, are further prefixed with an underscore ("_"). For example, SP interface functions are prefaced with "sp_".

[2] The volume checksums need not be cryptographically sealed since `SRH` protects them.

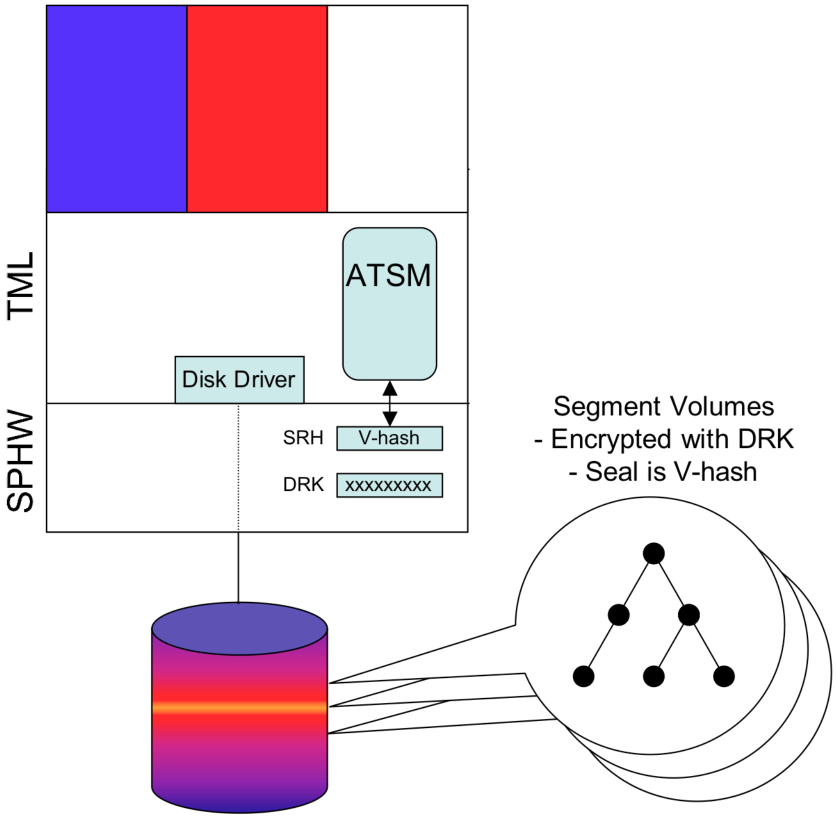

Figure 1. Segment Volume Protection

2. Storage Procedures

Storage procedures utilize the cryptographic transformation of `sp_secure_store` to efficiently encrypt data before it is written to disk. Encryption of user data can be handled in different ways. For example, the `tml_disk_flush` function can be exported to applications that are segment aware, or the TML could coerce any calls to write to the disk to use this logic.

To encrypt a segment on disk, applications (or modules within TML) call `tml_secure_flush` with the handle of the segment. `tml_disk_flush` first ensures that all of the "cache lines" of the segment are marked for encryption by calling `_tml_mark_secure`, and then calls `lpsk_flush2disk` to write the encrypted segment to disk.

```
tml_disk_flush(user_seg: evol_seg_handle) = (
      for i = 1 .. Length(user_seg) do
            if not user_seg.i.SecureData then  // secure data tag bit
                  _tml_mark_secure(user_seg.i, user_seg, i)
      lpsk_flush2disk(user_seg)
```

`_tml_mark_secure` is handed a word of data and a memory destination (segment and offset). It moves the word into a general purpose register and then calls `_atsm_secure_store` to mark the cache line for encryption.

```
_tml_mark_secure(w: word, dest_seg: ivol_seg_handle, offset: word) = (
        hw_move(R2, w)
        _atsm_secure_store(dest_seg, offset, R2)) // w encr with DRK
```

The internal procedure `_atsm_secure_store` calls `sp_secure_store` to mark the cache line for encryption:

```
_atsm_secure_store(dest_seg, offset, R2) = (
        sp_secure_store(dest_seg, offset, R2))
```

`lpsk_flush2disk` pushes to memory the elements of the segment that are in the processor cache, which causes SP to encrypt them, and then writes the entire segment to the disk. If the operation is called from within an ATSM, a DMA disk device must be used to copy the encrypted segment directly from memory onto the disk,[vii] because using the processor to write to disk would cause the memory to be decrypted first - i.e., by pulling it back into the processor in order to write to the disk. If not in a TSM, then *programmed I/O* or DMA I/O can be used to write to the disk.

```
lpsk_flush2disk(user_seg) = (
        for i = 1 .. Length(user_seg) do
                x86_clflush(i) //flush all cache lines of segment
        _dma_device_write(user_seg))
```

D. Trusted Channels

Trusted channels are cryptographically-secured communication paths between a partition or a trusted component of the SC_Device and a remote entity such as the Authority or Third Party. Trusted channel set up is accomplished during initialization, at least during Phase II, to keep the runtime system simple. The channel keys and other sensitive channel parameters are stored in a TML database and are not available to applications; others parameters are made known to applications as necessary (e.g., the device handle). The TML encrypts the database with a key derived from the DRK. TML can recreate this key using its ATSM functions, so the key does not need to be stored in persistent memory. TML makes logical devices for accessing trusted channels available to partition subjects via a virtual BIOS interface. This mechanism can be made more efficient in the emergency partition, since the SCOS does not necessarily need to be presented with a legacy BIOS interface.

Activity on a trusted channel is initiated by outgoing communication from the device, during which the channel is mutually authenticated.

Table 1 shows the cryptographic keys[viii] for security functions (attestation, authentication, communication) between Authority, Third Party and the SC_Device. These keys are used to generate related session keys, etc.

For example, a trusted component of the SC_Device may establish a trusted channel with the Authority. The channel security is based on a session key generated with the `sp_derive` instruction from a non-secret seed that is shared with the Authority; since the DRK utilized by `sp_derive` is secret, the generated key will be secret.

Table 1. Authentication and Communication Security

Party 1	Party 2	Key
Authority	SC_Device Trusted Component	DRK
Authority	Third Party	Keys agreed to in SLA/MOU
Third Party	SC_Device Normal Partition	Keys provided by Third Party and installed at Depot

E. Hardware Support for Emergency State Management

The Authority maintains the global emergency state of the network and communicates state changes to SC_Devices. Trusted components of SecureCore (such as the Emergency Manager, "e-manager," and TPA[ix] in Figure 2) receive, store, and respond to emergency status updates. The Authority Mode SP processor is extended to enhance the assurance of emergency state management with:

- Two new instructions: `sp_update_emergency` and `sp_get_emergency`
- A local emergency-state counter: `e_counter`
- An emergency-state bit that is the local version of emergency state: `e_state`

The Authority keeps a record of the `sc_device_id` and `DRK` for each SC_Device. It also maintains the emergency state-change number.

1. Starting an Emergency State Change

To announce a change to the emergency state, the Authority first updates its own information (in this case, the start of an emergency):

```
emergency_counter = emergency_counter + 1
emergency_state = 1  // emergency is "on"
```

Then the Authority generates an emergency message for one or more SC_Devices,[x] and sends them each over a trusted channel (see Section IV.D). The message contains two parts, a payload and the hash of the payload.

The payload contains the new emergency state and the global state-change number; it is encrypted with each SC_Device's DRK (in this case, device number `n)`:

payload = _authority_encrypt(e-state, e-counter, DRK_n)

The crypto-hash[3] of the encrypted payload is also based on the target SC_device's DRK:

```
hash = _authority_keyed_hash(payload, DRKn)
```

The message is the concatenation of the payload and the hash:

```
message = _authority_concatenate(payload, hash)
```

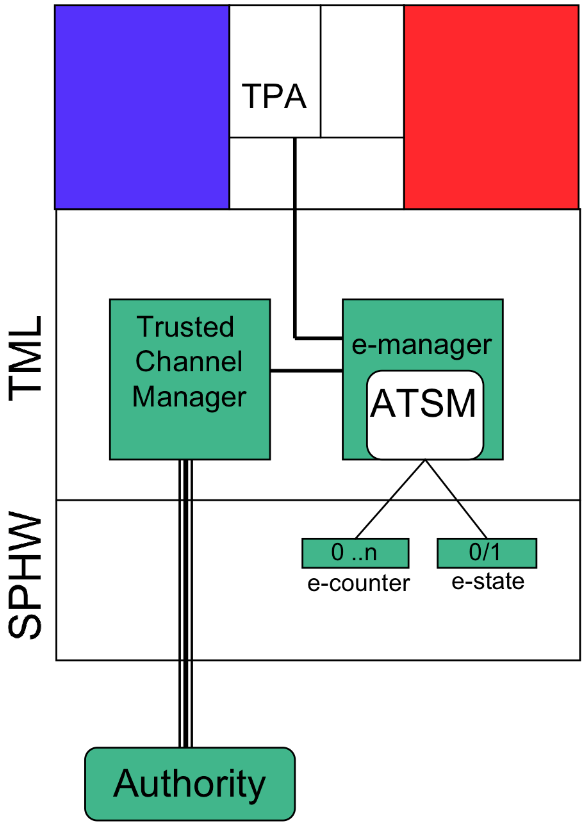

Figure 2. Local Emergency-state Management

An SCSS trusted channel manager on the SC_Device receives the message, and sends it to the e-manager. The e-manager enters A-CEM and invokes `sp_update_emergency(message)` to update the emergency state.

2. New SP Processing

SP processes `sp_update_emergency` with the following actions (see also Table 2), which appear from the interface to the hardware to be *atomic* (internal logical functions that are not available at the SP interface are prefaced with an underbar "_"). Before processing, SP ensures that the calling program has sufficient privilege.

[3] This hash function is distinct from the function used by `sp_derive`, to help prevent replay attacks (see Section IV.E.4).

```
      sp_update_emergency(message) =
            tmp := _sp_check_hash(DRK, message.payload, message.hash)
            if tmp = "success" then
                  tmp := _sp_decrypt(DRK, message.payload, tmp_values)
                  if tmp = "success" then
                        if tmp_values.counter > e_counter then
                              e_counter := tmp_values.counter
                              e-state := tmp_values.state
                        else tmp = "bad_ecounter"
            return (tmp)  // "success" or a specific error
```

`sp_update_emergency` validates the hash[3] against the payload using the SP_Device's DRK, and then decrypts the payload with the DRK. It also checks that the new counter value is greater than the previous value, to ensure that SP has not been requested to perform duplicate updates. If these operations are successful, SP writes the decrypted payload state to the hardware `e-state` register.

Table 2. Emergency HW Instructions

Instruction	Arguments	Exceptions

`sp_update_emergency`	*`subject`*`: implicit`
	`message: string`
		`Subject lacks sufficient privilege`
		`Message has wrong format`
		`The message counter value is too low`
		`The message hash does not match`

`sp_get_emergency`	*`subject:`*` implicit`
	`GR1: word`	

The calling subject shown is an implicit parameter, which indicates the security attributes of the program or module that invokes the instruction.
Privileged domains are those less than ring 0 for VTx; less than ring 1 otherwise.

3. Responding to State Change

If `sp_update_emergency` was successful, the e-manager sends an acknowledgement to the Authority and calls `sp_get_emergency`:

```
      sp_get_emergency  = return (e-state)
```

The e-manager takes appropriate action depending on whether an emergency has started or stopped. In the former case, it will signal to TPA to announce an emergency to the user and make the Emergency Partition available; or it will initiate the termination of an existing emergency.

4. Emergency State Threats and Security Analysis

The correct management of emergencies depends on the correct interpretation of the global emergency state by SC_Devices. Threats to this correctness include corrupted, impersonated or disrupted communication of the emergency message, and corrupted, impersonated or disrupted processing or storage of the emergency state on the local SC_Device. Additionally, in situations where confidentiality of emergency state changes is required, eavesdropping of emergency communication, storage, and processing is a concern.

COMMUNICATIONS INTEGRITY

It does not appear that the uniqueness of the emergency message can be ensured in a simple one-way communication from the Authority to the SP hardware, as it would be difficult for the SC_Device to distinguish previous authentic messages that it had missed from authentic current messages.

The use of the trusted channel ensures that the emergency message is from a trusted source and is current. The emergency hash and counter provide additional defense-in-depth that the message is authentic, current, and unmodified during communication.

COMMUNICATIONS AVAILABILITY

Availability is problematic, as we cannot guarantee connectivity of the SC_Devices, but the SC_Devices are required to use best effort to stay synchronized with the Authority.

An SC_Device may be offline occasionally, for example during extreme emergency conditions, but the proposed emergency policy allows the user to continue to have access to emergency data if the SC_Device loses connectivity during an emergency.[xi] During the offline period, it is conceivable the emergency could have cycled on and off one or more times. The individual SC_Device does not need to be concerned with offline state changes; only that it should re-synchronize its emergency state before any further user activity is allowed. Upon reconnection to the network, before any user activities are allowed, the TML must request an emergency state packet from the Authority in order to synchronize with the current emergency state.

STORAGE AND PROCESSING INTEGRITY

On the SC_Device, emergency state is managed exclusively by trusted components. State changes are validated and stored in hardware, and access to the hardware state is restricted to ATSMs. The hardware support enhances SecureCore protection against attacks on the storage and processing of emergency state and reduces the amount of software that needs to be trusted for emergency state management.

One potential threat is that *corrupted trusted software* (while the SecureCore security architecture provides high assurance self-protection, this analysis is provided to support defense-in-depth) on the SC_Device could sign messages with the DRK by using `sp_derive`, and then spoof the SC_Device into believing the wrong emergency status. However, protection against these counterfeit emergency messages is accomplished by the use of a different emergency-signing algorithm[xii] than that used by `sp_derive` (e.g., the signatures could be a different length).

EMERGENCY CONFIDENTIALITY

Confidentiality is ensured through several means: the use of a trusted channel for communications, the use of trusted components for management of emergency state , and SP hardware support for hiding the content of emergency messages during processing and storage (which even hides the value of upcoming state changes from the trusted components until after the hardware state change has occurred).

V. References

[1] Boneh, D. and Franklin, M. “Identity based encryption from the Weil pairing.” *SIAM J. of Computing*, Vol. 32, No. 3, pp. 586-615, 2003.

[2] Clark, Paul C., Irvine, Cynthia E., Levin, Timothy E., Nguyen, Thuy D., Vidas, Timothy M., SecureCore Software Architecture: Phased Implementation, NPS-CS-0n-nnn, Naval Postgraduate School, to appear circa 2007.

[3] Davis, R. *Government Records - Results of a Search for Records Concerning the 1947 Crash Near Roswell, New Mexico.* GAO/NSIAD-95-187. Director, National Security Analysis. July 1995 http://www.fas.org/sgp/othergov/roswell.html (PDF Version) – last accessed October 3, 2007.

[4] Denning, D. “A Lattice Model of Secure Information Flow.” *Communications of the A.C.M.* Vol. 19 No. 5. pp. 236-243. 1976

[5] Dwoskin, J. and Lee, R. “Hardware-rooted Trust for Secure Key Management and Transient Trust.” CCS’07, October 29–November 2, 2007, Alexandria, Virginia, USA.

[6] Irvine, C. Collaborative Research: SecureCore for Trustworthy Commodity Computing and Communications. 31 Mar. 2005. https://www.fastlane.nsf.gov/servlet/showaward?award=0430566

[7] Irvine, C. “A Multilevel File System for High Assurance.” Proceeding of the IEEE Symposium on Security and Privacy. IEEE Computer Society Press. pp. 78-87. Oakland, CA May-95.

[8] Levin, T., Irvine, C., Weissman, C., Nguyen, T., "Analysis of Three Multilevel Security Architectures" to appear in proceedings of the Computer Security Architecture Workshop, ACM. November 2, 2007, Fairfax, Virginia, USA. http://www.cisr.nps.navy.mil/pub_papers.html

[9] Levin, T. *SP Summary (with Authority Mode)*. SecureCore Project Working Note. September 18, 2007.

[10] Reed, D. and Kanodia, R. “Synchronization with Eventcounts and Sequencers.” *Communications of the A.C.M.* Vol. 22 pp. 115-123, 1979.

Appendix A: Notes on Design Considerations and Future Work

Labeling of the segments in the internal volumes can be per-object, or the volumes can be partitioned per access class with a root for each class. The latter approach would save space but would dis-allow "multilevel directories" containing objects at different levels (all dominating their directory level), but *soft links* can give the appearance of MLS.[7]

A. SP

Consider whether it would make sense for `sp_secure_store,` when invoked by a UTSM (vice Authority Mode), to be based on the `UMK` rather than `DMK`, so that the user's data could be portable to another of his or her machines – assuming `sp_secure _store` is used for persistent storage.

It may be possible to use `DRK` as private key in a public key encryption algorithm,[1] if it is paired with a public key, e.g., accessible via the Authority web site. In this usage, `DRK` would be limited to "signing" 2-word values by ATSMs.

B. Considerations cited in the text

[i] Emergency state is binary in Phase II, but can be expanded, for example, to a scale of emergency conditions in the future. Also, in Phase II, emergency partitions may exist at different sensitivity levels for different Third Parties; and groups of Third Parties may share an Emergency Partition if they so choose.

[ii] Support for information-rich emergency announcements is left for future work.

[iii] Read down and write up across the network may be considered.

[iv] In later phases of SecureCore, multiple third parties will be supported. Third parties who trust each other can be co-located in the same emergency partition; however, third parties who trust no other Third Party will be isolated in their own emergency partition – these operational arrangements follow from the observation that if information from two equivalence classes is allowed to intermingle, then they are in effect a single equivalence class (e.g., see Denning's discussion of irredundant classes[4]).

[v] We will consider providing additional support for receiving information from the Third Party, whereby the information can be received by a trusted application (in the trusted partition) that then places the data in the emergency partition.

[vi] Recovery of segments and volumes, e.g., in the event of an unexpected shutdown, will be addressed in the future. For example, segments may have their own seals. The volume seal may be a function of the seals of the segments on the volume. In order to avoid re-sealing the volume on every segment change, the volume seal could be allowed to be out of date with respect to some (small) set of segments, e.g., if the TML stores the old version of those segments that were sealed since the previous volume seal. The TML could recover to a secure state comprising the sealed volume plus a set of sealed segments. Another option is to avoid volume seals and just seal the segments.

[vii] As an alternative to using CLFLUSH and a DMA disk device to encrypt persistent storage, a new SP instruction could provide "secure write" capability, so that the kernel could write encrypted data to memory or devices.

[viii] A PKI could be used for distribution of keys for normal- and emergency-partition channel trusted channels – an engineering consideration.

[ix] Functions of the e-manager and TPA might be provided by multiple modules in the future.

[x] Broadcast of changes in emergency state to groups of SC_Devices, and peer-to-peer shuttling of this information, may be investigated in the future.

[xi] We will consider configuration options to modify SC_Device's response to (its own perception of) lack of connectivity to the Authority. For example, a given Authority or Third Party may desire the SC_Device to discontinue an emergency after it has been disconnected for a certain period.

[xii] An alternative to using a different signing algorithm for emergency processing than the one used for sp_derive is to add a dedicated, persistent-memory SP register to hold an emergency integrity-checking key. This key would be inaccessible to software through any HW instruction; rather, the sp_update_emergency instruction would implicitly use this key for validation of the emergency message. The tradeoff here is the complexity of an additional dedicated register, versus the complexity of an additional signing algorithm.

This page left intentionally blank

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center 1
   Ft. Belvoir, VA

2. Dudley Knox Library 1
   Naval Postgraduate School
   Monterey, CA

3. Karl Levitt 1
   National Science Foundation
   4201 Wilson Blvd.
   Arlington, VA

4. Lee Badger 1
   DARPA
   Arlington, VA

5. Timothy E. Levin 2
   Naval Postgraduate School
   Monterey, CA

6. Ganesha Bhaskara 1
   Information Sciences Institute
   University of Southern California
   Marina del Rey, Ca

7. Thuy D. Nguyen 2
   Naval Postgraduate School
   Monterey, CA

8. Paul C. Clark 2
   Naval Postgraduate School
   Monterey, CA

9. Terry V. Benzel 1
   Information Sciences Institute
   University of Southern California
   Marina del Rey, Ca

10. Cynthia E. Irvine 2
    Naval Postgraduate School
    Monterey, CA

This page left intentionally blank